Sorting Comparisons

We have discussed five different sorting algorithms: *Bubble Sort, Insertion Sort, Selection Sort, Merge Sort,* and *Quick Sort.* You do **not** have to implement these algorithms; instead, you will write a driver that will sort various lists using these algorithms, calculating their execution time, and printing a table of results.

The file sort_method.txt contains:

- Some import statements and settings that your program will likely need.
- Function definitions for the five sorting algorithms listed above, plus any "helper" functions used by them.

Your program should do the following:

1. Generate a list containing n integers in *random* order.
2. Start a timer.
3. Use the bubbleSort() function to sort the list.
4. Stop the timer.
5. Repeat steps 1-4 five times, and calculate an average time.
6. Repeat steps 1-5 three times: once for n=10, once for n=100, and once for n=1000.
7. Repeat steps 1-6 for each of the five sorting algorithms.
8. Save all of the average times from the above test cases, and print them out in a table. (See the "output" section below.)

Repeat the above process, but using *sorted* lists instead.

Repeat the above process, but using lists sorted in *reverse* order instead.

Repeat the above process, but using lists that are *almost sorted* instead. Generate a sorted list, and then swap *n* pairs of elements randomly, where *n* = 10% of the length of the list. (So for list length 100, you should make 10 random swaps.)

Hints:

To calculate the runtime of a piece of code, use the method time.perf_counter() (for Python 3) or time.time() (for Python 2). A call to this method returns the current time (down to many decimal places of accuracy). To time a piece of code (such as a function call to bubbleSort(), take snapshots of the clock immediately before and after the code:

```
# Python 2 version
startTime = time.time()
bubbleSort(myList)
endTime = time.time()
elapsedTime = endTime - startTime

# Python 3 version
startTime = time.perf_counter()
```

```
bubbleSort(myList)
endTime = time.perf_counter()
elapsedTime = endTime - startTime
```

To generate a sorted list of integers, use a list comprehension:

```
myList = [i for i in range(listLength)]
```

To randomize the order of the elements in an entire list, use the method random.shuffle(). (Note that shuffle does not return anything, it randomizes the list *in place*.)

```
random.shuffle(myList)
```

To generate a random integer between integers a and b inclusive, use the method random.randint(). To randomly select the index of an element in the list myList, you could say:

```
randomIndex = random.randint(0,len(myList))
```

Input:

There is no input file for this assignment. You will write your own code to generate lists of length 10, 100, and 1000, in either random, sorted, or reverse-sorted order, to use as data for your program.

Output:

The output from your program should closely resemble the following. (The numbers below are completely made up, so don't expect to get the same results: your *format* should be identical to the below, but everyone will get different *values* for the numbers.)

```
Input type = Random
                    avg time   avg time   avg time
   Sort function     (n=10)     (n=100)    (n=1000)
-------------------------------------------------------
      bubbleSort    0.000107   0.003255   0.311164
   selectionSort    0.000073   0.001299   0.137412
   insertionSort    0.000062   0.001458   0.170794
       mergeSort    0.000101   0.001004   0.026353
       quickSort    0.000137   0.000986   0.011964


Input type = Sorted
                    avg time   avg time   avg time
   Sort function     (n=10)     (n=100)    (n=1000)
-------------------------------------------------------
      bubbleSort    0.000073   0.001472   0.224019
   selectionSort    0.000070   0.001367   0.142006
   insertionSort    0.000048   0.000096   0.000622
       mergeSort    0.000096   0.000804   0.011235
       quickSort    0.000085   0.001798   0.177582
```

```
Input type = Reverse
                    avg time   avg time   avg time
   Sort function     (n=10)     (n=100)    (n=1000)
-----------------------------------------------------
      bubbleSort    0.000086   0.003821   0.431074
   selectionSort    0.000070   0.001333   0.134598
   insertionSort    0.000072   0.002678   0.303098
       mergeSort    0.000100   0.000846   0.011272
       quickSort    0.000153   0.003479   0.206156

Input type = Almost sorted
                    avg time   avg time   avg time
   Sort function     (n=10)     (n=100)    (n=1000)
-----------------------------------------------------
      bubbleSort    0.000083   0.003829   0.431275
   selectionSort    0.000072   0.001334   0.132591
   insertionSort    0.000065   0.002682   0.308093
       mergeSort    0.000115   0.000854   0.012271
       quickSort    0.000123   0.003477   0.208143
```

Remember that the numbers shown are the average runtime for each test, averaged over five trials.